AMENDMENTS TO THE CLAIMS

1. (Currently Amended) A method comprising:

building a control flow graph (CFG) for a loop body of a sequential application program to form a CFG loop;

updating nodes of the CFG loop to enclose identified critical sections of the sequential application program within corresponding pairs of boundary instructions; and

modifying the updated nodes of the CFG loop to reduce an amount of instructions between the corresponding pairs of boundary instructions to form a modified CFG loop;

generating a plurality of program-thread-partitions from the modified CFG loop;

concurrently executing ~~a~~ the plurality of ~~application~~ program-thread-partitions that are generated from the modified CFG loop; and

synchronizing execution of the identified critical sections of the sequential application program among the plurality of concurrently executing ~~application program threads~~ program-thread-partitions to ensure that the identified critical sections are executed in a sequential thread order.

2. (Original) The method of claim 1, wherein updating the CFG loop comprises:

selecting an identified critical section of the sequential application program;

inserting an await instruction within a top node of the CFG loop;

inserting an advance instruction within a bottom node of the CFG loop; and

repeating the selecting, inserting and inserting for each identified critical section of the sequential application program.

3. (Previously Presented) The method of claim 1, wherein modifying the updated nodes of the CFG loop comprises:

hoisting detected hoist instructions from identified motion candidate instructions within the nodes of the CFG loop using code motions with fixed await boundary instructions into corresponding predecessor basic blocks;

sinking detected sink instructions from identified motion candidate instructions within the nodes of the CFG loop using code motion with fixed advance instructions into corresponding successor basic blocks; and

reordering identified motion candidate instructions within the nodes of the CFG loop with fixed await instructions and fixed advance instructions.

4. (Previously Presented) The method of claim 3, wherein hoisting detected hoist instructions with fixed await instructions comprises:

identifying every instruction within a basic block the CFG loop, excluding await instructions, as a motion candidate instructions;

building an inverse graph of the CFG loop;

initializing a hoist queue with the basic blocks from the CFG loop, the basic blocks ordered according to a topological order indicated by the inverse graph;

hoisting motion candidate instructions of the basic blocks into corresponding predecessor basic blocks until hoist instructions are no longer detected from the identified motion candidate instructions; and

hoisting detected hoist instructions from motion candidate instructions within a source basic block of the CFG loop according to a dependence graph of the sequential application program.

5. (Previously Presented) The method of claim 4, wherein hoisting detected hoist instructions from the motion candidate instructions of the basic blocks comprises:

de-queuing a basic block from the hoist queue as a current block;

computing hoist instructions from the motion candidate instructions based on a dependence graph of the sequential application program;

hoisting the computed hoist instructions into a corresponding basic block; and

enqueuing the current block's predecessors from the CFG loop into the hoist queue when a change is detected.

6. (Previously Presented) The method of claim 4, wherein sinking detected sink instructions with fixed advance instructions comprises:

identifying motion candidate instructions within the basic blocks of the CFG loop through dataflow analysis with fixed advance instructions;

initializing a sink queue with the basic blocks ordered based on a topological order in the CFG loop;

sinking detected sink instructions among the basic blocks into corresponding successor basic block until sinking instructions are no longer detected from the identified motion candidate instructions; and

reordering detected motion candidates within basic blocks that contain advance instructions according to the dependence graph.

7. (Previously Presented) The method of claim 6, wherein sinking detected sink instructions among the basis blocks comprises:

de-queue a basic block from the sink queue as a current block;

computing sink instructions from motion candidate instructions based on a dependence graph of the sequential application program;

sinking computed sink instructions into a corresponding successor basic block; and

en-queuing a current block's successors in the CFG loop into the sink queue if a change is detected.

8. (Previously Presented) The method of claim 3, wherein performing instruction hoisting with both the await instructions and advance instructions fixed, comprises:

initializing a hoist queue with the basic blocks ordered based on a topological order in the CFG loop;

identifying motion candidate instructions within the basic blocks of the CFG loop through dataflow analysis with fixed advance instructions and fixed await instructions;

hoisting detected hoist instructions among the basic blocks into corresponding predecessor basic blocks until hoist instructions are no longer detected from the identified motion candidate instructions; and

reordering the identified motion candidate instructions within basic blocks that contain await instructions based on a dependence graph of the sequential application program.

9. (Original) The method of claim 8, wherein motion candidate instructions hoisted out of an outmost await instruction are no longer treated as motion candidates; and

wherein motion candidate instructions out of an outmost advance instruction are no longer treated as motion candidates.

10. (Cancelled.)

11. (Currently Amended) An article of manufacture including a machine readable medium having stored thereon instructions which may be used to program a system to perform a method, comprising:

building a control flow graph (CFG) for a loop body of a sequential application program to form a CFG loop;

updating nodes of the CFG loop to enclose identified critical sections of the sequential application program within corresponding pairs of boundary instructions; and

modifying the updated nodes of the CFG loop ~~are modified~~ to reduce an amount of instructions between corresponding pairs of boundary instructions to form a modified CFG loop;

generating a plurality of program-thread-partitions from the modified CFG loop;

concurrently executing ~~a~~ the plurality of ~~application~~ program-thread-partitions that are generated from the modified CFG loop ; and

synchronizing execution of the identified critical sections of the sequential program among the plurality of concurrently executing ~~application program threads~~ program-thread-partitions to ensure that the identified critical sections are executed in a sequential thread order.

12. (Original) The article of manufacture of claim 11, updating the CFG loop comprises:

selecting an identified critical section of the sequential application program;

inserting an await instruction within a top node of the CFG loop;

inserting an advance instruction within a bottom node of the CFG loop; and

repeating the selecting, inserting and inserting for each identified critical section of the sequential application program.

13. (Currently Amended) The article of manufacture of claim 11, wherein generating ~~forming~~ the ~~thread application partitions~~ program-thread-partitions comprises:

hoisting identified motion candidate instructions within the nodes of the CFG loop using code motions with fixed await boundary instructions;

sinking identified motion candidate instructions within the nodes of the CFG loop using code motion with fixed advance instructions; and

hoisting identified motion candidate instructions within the nodes of the CFG loop with fixed await instructions and fixed advance instructions.

14. (Previously Presented) The article of manufacture of claim 13, wherein hoisting detected hoist instructions with fixed await instructions comprises:

identifying every instruction within a basic block the CFG loop, excluding await instructions, as a motion candidate instructions;

building an inverse graph of the CFG loop; initializing a hoist queue with the basic blocks from the CFG loop, the basic blocks ordered according to a topological order indicated by the inverse graph;

hoisting motion candidate instructions of the basic blocks until hoist instructions are no longer detected from the motion candidate instructions; and

reordering detected hoist instructions from motion candidate instructions in a source basic block of the CFG loop according to a dependence graph of the sequential application program.

15. (Original) The article of manufacture of claim 14, wherein hoisting detected hoist instructions from the motion candidate instructions of the basic blocks comprises:

de-queuing a basic block from the hoist queue as a current block;

computing hoist instructions from the motion candidate instructions of the basic blocks based on a dependence graph of the sequential application program;

hoisting the computed hoist instructions into a corresponding basic block; and

enqueuing the current block's predecessors from the CFG loop into the hoist queue when a change is detected.

16. (Original) The article of manufacture of claim 14, sinking detected sink instructions with fixed advance instructions comprises:

identifying motion candidate instructions within the basic blocks of the CFG loop through dataflow analysis with fixed advance instructions;

initializing a sink queue with the basic blocks ordered based on a topological order in the CFG loop;

sinking detected sink instructions among the basic blocks until sinking instructions are no longer detected; and

sinking detected motion candidates within basic blocks that contain advance instructions according to the dependence graph.

17. (Original) The article of manufacture of claim 16, wherein sinking detected sink instructions among the basis blocks comprises:

de-queue a basic block from the sink queue as a current block;

computing sink instructions from motion candidate instructions based on a dependence graph of the sequential application program;

sinking computed sink instructions into a corresponding basic block; and

en-queuing a current block's successors in the CFG loop into the sink queue if a change is detected.

18. (Original) The article of manufacture of claim 13, wherein performing instruction hoisting with both the await instructions and advance instructions fixed, comprises:

initializing a hoist queue with the basic blocks ordered based on a topological order in the CFG loop;

identifying motion candidate instructions within the basic blocks of the CFG loop through dataflow analysis with fixed advance instructions and fixed await instructions;

hoisting detected hoist instructions among the basic blocks until hoist instructions are no longer detected; and

hoist motion candidates within basic blocks that contain await instructions based on a dependence graph of the sequential application program.

19. (Original) The article of manufacture of claim 18, wherein motion candidate instructions hoisted out of an outmost await instruction are no longer treated as motion candidates; and

wherein motion candidate instructions out of an outmost advance instruction are no longer treated as motion candidates.

20. (Cancelled.)

21. (Currently Amended) A method comprising:

partitioning a sequential application program into a plurality of ~~application program threads~~ program-thread-partitions according to a predetermined thread count of a multi-threaded architecture, each program-thread-partition including a program-thread-partition loop;

concurrently executing the plurality of ~~application program threads~~ program-thread-partitions within respective threads of the multi-threaded architecture; and

synchronizing access to identified critical sections of ~~thread program~~ program-thread-partition loops of the sequential application program among the plurality of concurrently executing program-thread-partitions ~~application program threads~~ to execute the identified critical sections of the ~~thread program~~ program-thread-partition loops in sequential thread order.

22. - 23. (Cancelled.)

24. (Currently Amended) The method of claim 21, wherein partitioning the sequential application program into the plurality of program-thread-partitions ~~application program threads~~ further comprises:

receiving an indication of the identified critical sections within the sequential application program;

~~generating the plurality of application program threads according to the thread count to synchronize access to the identified critical sections among the thread program loops;~~ and

processing the identified critical sections to reduce an amount of code contained within the critical sections of the ~~thread program~~ program-thread-partition loops.

25. (Currently Amended) The method of claim 24, wherein code motion is used to reduce

the amount of code contained within the identified critical sections of the ~~thread program~~ program-thread-partition loops.

26. (Currently Amended) An article of manufacture including a machine readable medium having stored thereon instructions which may be used to program a system to perform a method, comprising:

partitioning a sequential application program into a plurality of ~~application program threads~~ program-thread-partitions according to identified critical sections within the sequential application program, each program-thread-partition including a program-thread-partition loop;

concurrently executing the plurality of ~~application program threads~~ program-thread-partitions within respective threads of ~~a~~ the multi-threaded architecture; and

synchronizing access to ~~the~~ identified critical sections among the plurality of concurrently executing ~~application program threads~~ program-thread-partitions to ensure that the identified critical sections of program-thread-partition loops are executed in a sequential thread order.

27. – 28. (Cancelled.)

29. (Currently Amended) The article of manufacture of claim 26, wherein ~~generating~~ partitioning the sequential application program ~~threads~~ comprises:

processing identified critical sections to reduce an amount of code contained within critical sections of ~~thread program~~ program-thread-partition loops.

30. (Currently Amended) The article of manufacture of claim 29, wherein code motion is used to reduce the amount of code contained within critical sections of the ~~thread program~~ program-thread-partition loops.

31. - 33. (Cancelled.)

34. (Currently Amended) A system comprising:

a processor;

a memory controller coupled to the processor; and

a DDR SRAM memory coupled to the processor, the memory including a compiler is operable to cause partitioning of a sequential application program into a plurality of ~~application program threads~~ program-thread-partitions according to identified critical sections within the sequential application program to enable concurrent execution of the plurality of ~~application program threads~~ program-thread-partitions within a respective thread of a multi-threaded architecture and to synchronize access to the identified critical sections among the plurality of concurrently executing ~~application program threads~~ program-thread-partitions to ensure that the identified critical sections are executed in a sequential thread order.

35. (Previously Presented) The system of claim 34, wherein the compiler is operable to cause building a control flow graph (CFG) for a loop body of a sequential application program to form a CFG loop to cause an update of nodes of the CFG loop to enclose identified critical sections of the sequential application program within pairs of boundary instructions and to cause modification of nodes of the CFG loop to reduce an amount of instructions between corresponding pairs of boundary instructions to form a modified CFG loop.

36. (Previously Presented) The system of claim 35, wherein the compiler is operable to cause hoisting identified motion candidate instructions within the nodes of the CFG loop using code motions with fixed await boundary instructions, to cause sinking of identified motion candidate instructions within the nodes of the CFG loop using code motion with fixed advance instructions and to cause hoisting of motion candidate instructions within the nodes of the CFG loop with fixed await instructions and fixed advance instructions.